15437-0541/P6523 *Patent*

UNITED STATES PATENT APPLICATION

FOR

DYNAMIC RECONFIGURATION OF APPLICATIONS ON A SERVER

INVENTOR(S):

ARVIND SRINIVASAN
MURTHY CHINTALAPATI

PREPARED BY:
HICKMAN PALERMO TRUONG & BECKER LLP
1600 WILLOW STREET
SAN JOSE, CALIFORNIA 95125-5106
(408) 414-1080

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EL558817065US

Date of Deposit November 26, 2001

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Box Patent Application, Commissioner of Patents, Washington, D.C. 20231.

Tirena Say
(Typed or printed name of person mailing paper or fee)

Tirena Say
(Signature of person mailing paper or fee)

DYNAMIC RECONFIGURATION OF APPLICATIONS ON A SERVER

Inventor(s): Arvind Srinivasan and Murthy Chintalapati

Cross Reference to Related Application

[0001] The present application is related to commonly owned U.S. Patent Application No. 09/792,805 entitled "Mechanism for Reconfiguring A Server Without Incurring Server Down Time," filed on February 23, 2001, which is herein incorporated by reference in its entirety.

Field of the Invention

[0002] This invention relates generally to computer systems, and more particularly to a technique for enabling server-based applications to be reconfigured without incurring server down time.

Background

[0003] With the continuing evolution of network computing, distributed applications utilizing a client/server computing model have become quite prevalent. Server-based, or web applications, have become a primary purveyor of distributed computing resources to client users connected to web servers and other servers through a network.

[0004] Applications that execute on a server require reloading and reinitialization when the underlying software code is revised or when the application configuration is revised. Furthermore, the server may be in the midst of processing requests on behalf of users for the revised application while the application is being reloaded or reinitialized.

[0005] One prior approach to reconfiguring a server-based application upon an application revision is to stop the server from processing, reload the revised application

and reconfigure the server accordingly, and then restart the server in order to implement the changes made to the application and the server. This approach has numerous shortcomings, one of which is the fact that the server cannot process requests for the revised application, or any other application installed on the server, while it is off-line or in the process of restarting. Requests that are in progress when the server is restarted may be interrupted, which typically leads to request processing errors. Furthermore, any server down-time adversely affects the availability of the server, which in turn may have an impact on, for example, SLAs (Service Level Agreement) associated with the services provided by the server.

[0006] Other shortcomings of this approach are due to the fact that large servers typically have multiple applications executing, thus an application maintenance operation requiring server stop/restart each time an application is changed is not a viable approach for a high-traffic server, partly due to the time it takes to restart a server running multiple applications. Again, while the server is off-line and while being restarted, users may be denied access to the server. A complex server can take on the order of minutes to restart. In terms of request traffic, several minutes is an extremely long time, and a tremendous amount of traffic can be lost in that time. Many servers, such as those servicing commercial websites, cannot afford to have this amount of down time. In addition, discovery of an error in the new application configuration subsequent to the reload requires another server stop/restart. Yet another shortcoming to note is that interrupting requests that are in progress by restarting the server likely compromises the integrity of the data associated with the current user sessions, which leads to errors when the sever resumes its request processing.

[0007] Another prior approach to reconfiguring a server-based application is to configure the server to periodically check the application class files for revisions thereto. Using this approach, the timestamp associated with each application class file is stored when starting the server and thus reloading the application. Upon an application service request, the server checks to see if a predetermined time interval has expired, and if it has, the server compares the timestamp of each class file stored on disk with the associated timestamp that the server had when it started. If the two timestamps are different, then the application class files are reloaded to ensure the that the current version of the application is being used to process requests.

[0008] This approach also has numerous shortcomings, one of which is that changes to the application configuration files, which are key application components, are not detected. This approach only detects changes to the application code, for example, the Java source code that is compiled into the application class files. Furthermore, any changes to the container, or runtime environment, in which the application is running, are also not detected. This is a bad situation because, typically, configuration changes to a container which directly or indirectly affect the application must cause the application to be reloaded and reinitialized so that the configuration used by the server reflects the current configuration of the application and the container. This approach does not facilitate such a process. Another shortcoming to this prior approach is that a server administrator typically waits for the predetermined time to elapse, and thus changes to the application are not detected until expiration of the time interval. Hence, one might have to wait to verify that the revised application is executing and functioning properly. Yet

another shortcoming is that each time the periodic check is performed, it produces a negative impact on server performance.

[0009] The current methodology for reconfiguring a server leaves much to be desired. In light of the foregoing, there is a clear need for an improved server-based application reconfiguration mechanism.

Summary of the Invention

[0010] In light of the shortcomings discussed above, the present invention provides an improved mechanism for reconfiguring an application that runs on a server, which enables the application to be reconfigured without incurring server down time and without interrupting application service requests or losing data associated with existing user sessions.

[0011] A method for reconfiguring an application running on a server, without restarting the server, include steps of reading application configuration information related to a new version of the application, constructing an application configuration based on the application configuration information, and providing the application configuration to the server. Advantageously, an application runtime environment within the server services new application service requests received on a new connection by referencing the new application configuration, while old application configurations are maintained for servicing, without interruption, existing service requests received on an existing connection.

[0012] According to one embodiment, the server determines whether the new application configuration has successfully initialized by receiving an indication from the runtime environment after consideration to the new application configuration, prior to changing a pointer to the revised application configuration in order to process requests on new connections. In another embodiment, persistent session information related to existing application user sessions is accessed to use in servicing new requests from the same user during the same user session.

[0013] As part of a server reconfiguration process, generally, server-based application reconfiguration is implemented as follows. At system startup, a server process is instantiated (the terms server and server process will be used interchangeably herein), and based upon a set of configuration information, the server constructs a set of runtime configuration data structures, including application configurations for applications installed on the server. These data structures are stored in a portion of the server process's memory space, and a reference or pointer to that portion of the memory space is stored in a global variable.

[0014] Once constructed, the configuration data structures are used to process incoming service requests. More specifically, in one embodiment, the server receives a request from a client for an application service. In providing the service, the server consults the global variable to obtain the pointer to the configuration data structures, and associates that pointer with the server connection, and thus the user session. Thereafter, all requests, events, and activities related to that application are processed by the server in accordance with the configuration data structures.

[0015] At some point, the application configuration information is revised (e.g. by an application developer) and the server receives a signal to reconfigure itself in accordance with the revised application configuration information. In response to this signal, the server constructs a new set of runtime configuration data structures based upon the modified application configuration information. In one embodiment, the server constructs the new data structures in much the same way as it did the first set of data structures, except that the new data structures are not constructed as part of a startup procedure, but rather in the background while continuing to process service requests.

Thus, to construct the new data structures, the server does not need to be shut down and restarted.

[0016] While the server is constructing the new set of data structures, the global variable continues to store a pointer to the first set of data structures; hence, all connections established by the server while the new data structures are being constructed are associated with the first set of data structures. Consequently, all requests, events, and activities received on those connections, including those related to the application, are processed by the server in accordance with the first set of data structures.

[0017] Once the new configuration data structures are constructed and stored by the server into another portion of the server process's memory space, the pointer value in the global variable is updated atomically. More specifically, the server changes the pointer value in the global variable such that it points to the new set of data structures rather than the first set of data structures. By doing so, the server in effect changes its configuration, thus changing the application configuration to the revised version of the application. That is, by changing the pointer in the global variable, the server causes all future application service requests received on new connections to be associated with the new set of data structures rather than the first set. This in turn causes all requests, events, and activities on those future service requests to be processed in accordance with the new data structures. Since the new data structures reflect the revised application configuration information, switching to the new data structures has the effect of reconfiguring the application. Pending service requests or new service requests on an old connection are still processed according to the first set of data structures. In this manner, a smooth

transition from the old application configuration to the new application configuration is achieved.

[0018] By eliminating the need for server down time, the present invention overcomes the shortcomings of the current methodology, and provides a significant advance in the art.

Brief Description of the Drawings

[0019] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0020] FIG. 1A is a partial block diagram illustrating a system in which an aspects of the present invention may be implemented;

[0021] FIG. 1B is a partial block diagram continuing from FIG. 1A, illustrating a system in which an aspects of the present invention may be implemented;

[0022] FIG. 1C is a block diagram illustrating an architecture of a server on which aspects of the present invention may be implemented;

[0023] FIG. 2A is a partial flowchart illustrating a method for reconfiguring an application that executes on a server or other computer system without restarting the server/computer system;

[0024] FIG. 2B is a partial flowchart, continuing from FIG. 2A, illustrating a method for reconfiguring an application that executes on a server or other computer system without restarting the server/computer system;

[0025] Fig. 3 shows a hardware block diagram of a computer system on which an embodiment of the invention may be implemented.

Detailed Description of Embodiment(s)

[0026] FUNCTIONAL OVERVIEW OF SERVER RECONFIGURATION

[0027] FIGS. 1A and 1B are block diagrams of a system 100 in which one embodiment of the present invention may be implemented, the system comprising a plurality of clients 102, a network 104, a server 106, and a storage 118. For purposes of illustration, the invention will be described below with reference to a web server 106. However, it should be noted that the invention is not so limited. Rather, the invention may be implemented in any type of server or computer system in which configuration information is used to define the behavior of a server and of an application that executes on the server.

[0028] As used herein, the term configuration information refers broadly to any information that defines the behavior of a server, and that is used by the server to determine its operation or behavior at runtime. Examples of configuration information include but are not limited to: (1) virtual server definitions, including the definition of virtual server addresses, the mapping of domain names to IP addresses, and the specification of connections between certain sockets and certain virtual servers; (2) access control and security information that specifies, for example, which entities are allowed to access which resources; (3) resource management information, such as how many threads (e.g. acceptor threads) are to be allocated at system startup; (4) information specifying how requests are to be processed, such as which application is to be invoked in response to a particular URL; (5) information specifying a source for content, such as the root directory for a particular domain name; (6) secure socket layer configuration information, such as the ciphers that are used to establish a connection and the digital certificates that

are used to certify authenticity of important materials; and (7) application configuration information, such as deployment descriptor information, application classes, and archive file libraries. These and other types of information may be specified as configuration information. In one aspect, the present invention is directed to application configuration information, and techniques for updating such information.

[0029] In system 100, each of the clients 102 may be any mechanism capable of communicating with the server 106, including but not limited to a computer running a browser program. The client 102 may communicate with the server 106 using any known protocol, including but not limited to HTTP and FTP, and the client 102 communicates with the server 106 via the network 104. The network 104 may be any type of network, including but not limited to a local area network and a wide area network such as the Internet. The network 104 may even be as simple as a direct connection. Any mechanism capable of facilitating communication between the client 102 and the server 106 may serve as the network 104.

[0030] In system 100, the server 106 is the mechanism responsible for providing most of the functionality of the system 100. More specifically, the server 106 receives application service requests from the clients 102, and performs whatever operations are necessary to service the requests. In one embodiment, the mechanism responsible for servicing client requests is the request processing module 108. Among other functions, the request processing module 108 establishes connections with the clients 102, and processes whatever requests, events, and activities are received on those connections. In carrying out its functions, the request processing module 108 relies upon several sets of information, including a pointer 112, which points to the current set of runtime

configuration data structures 116(1), including the application configuration 146(1). The pointer 112, which in one embodiment is stored as a global variable, enables the request processing module 108 to access the current set of configuration data structures 116(1), and the data structures 116(1) identify to the request processing module 108 the configuration information that a container 114 uses to load and execute an application to process and respond to client requests. As will be explained in greater detail below, the container 114 relies on application configuration information 146 constituent to the configuration data structures 116 to run the requested application.

[0031] In addition to the request processing module 108, the server 106 further comprises a configuration manager 110. In one embodiment, it is the configuration manager 110 that constructs and maintains the runtime configuration data structures 116 and the pointer 112. The configuration manager 110 manages the data structures 116 and the pointer 112 at system startup time, upon a system reconfiguration request, and during normal operation.

[0032] More specifically, at system startup time, the configuration manager 110 constructs the initial set of runtime configuration data structures 116(1) based upon the configuration information 120 stored in the storage 118. The configuration data structures 116(1) are effectively an internal server representation of the configuration information 120. These data structures 116(1) are easy and convenient for the request processing module 108 to access and to manipulate at runtime. Once constructed, the configuration data structures 116(1) may be used by the server 106 to determine its behavior at runtime. In addition, container 114 utilizes application configuration information 146(1) to determine, locate, and apply executable application components for

servicing requests. To enable the request processing module 108 to access the initial set of data structures 116(1), the configuration manager 110 updates the pointer 112 to point to the data structures 116(1). Initial setup of the application's configuration is thus achieved.

[0033] During normal operation, the configuration manager 110 may receive a signal (e.g., from a system administrator or application developer) to reconfigure the server 106. More specifically, the configuration manager 110 may receive a signal to construct a new set of runtime configuration data structures based upon a modified set of configuration information 124. The modified set of configuration information 124 may comprise modified application configuration information 144 for applications deployed on the server 106. In response to such a signal, the configuration manager 110 accesses the set of modified configuration information 124 from the storage 118, and constructs a new set of runtime configuration data structures 116(2) based upon the modified configuration information 124. After the new data structures 116(2) are fully constructed, the configuration manager 110 updates the current configuration pointer 112 to point to the new set of configuration data structures 116(2) instead of the initial set of data structures 116(1). Hence, listener threads that are accepting new application service requests are then directed to the new set of configuration data structures 116(2), which in turn direct worker threads to execute the application of interest according to the current data structures 116(2). In one embodiment, this update operation is performed atomically (e.g. by using a lock) to prevent errors that may arise from other threads accessing the pointer 112 while it is being updated.

[0034] The container 114 provides the runtime environment for running applications on the server 106. In other words, it provides the resources, or at least access to the resources, required to process application service requests. By updating the current configuration pointer 112, the configuration manager 110 causes all server components (including the request processing module 108 and the container 114) that rely upon the pointer 112, to access the new data structures 116(2) instead of the initial set of data structures 116(2). By changing the pointer 112, thereby changing the set of configuration data structures the container 114 relies upon to run a particular application, the configuration manager 110 in effect changes the configuration of the applications running on the server 106. As will be described in greater detail in a later section, reconfiguring the server 106 in this manner causes the request processing module 108 to associate future connections with the new data structures 116(2) rather than the initial set of data structures 116(1).

[0035] Several points should be noted at this juncture. First, note that in constructing the new runtime configuration data structures 116(2), the server 106 does not need to be shut down and restarted. Rather, the configuration manager 110 constructs the new data structures 116(2) as part of its normal operation. In addition, note that at certain points in time, there may be more than one set of runtime configuration data structures 116 used by the container 114. Hence, some application service requests, such as requests pending on an existing connection at the time of reconfiguring the server, may be processed using one or more sets of configuration data structures (e.g., 116(1) and 146(1)), while new requests received on a new connection may be processed using another set of configuration data structures(e.g., 116(2) and 146(2)). Server 106 allows for this

possibility. In fact, this aspect of the server 106 enables a smooth transition to be achieved from one application configuration to another. In FIG. 1A, for the sake of simplicity, only two configuration data structures 116 are shown. It should be noted, though, that any number of configuration data structures 116, and thus application configuration information 146, may exist in the server's memory space at any time.

[0036] In one embodiment, the server 106 is implemented in a multi-threaded environment. In such an environment, the request processing module 108 and the configuration manager 110 may be running on different threads, which means that they may run concurrently. As a result, while the configuration manager 110 is constructing a new set of configuration data structures 116(2), the request processing module 108 may continue to receive connection requests and application service requests. The resultant connections and service responses will be associated with the initial set of configuration data structures 116(1), until the current configuration pointer 112 is updated to point to the new set of configuration data structures 116(2). Until this time, the new data structures 116(2) are not used by the request processing module 108 or any other server component. This helps to prevent errors and to facilitate a smooth transition from one configuration to another.

[0037] FUNCTIONAL OVERVIEW OF APPLICATION RECONFIGURATION

[0038] Methods for reconfiguring an application executing on a computer system without restarting the computer system are described. In one embodiment, implementations of the methods involve web applications running on a web or application server.

[0039] Referring again to FIGs. 1A and 1B, configuration information 120 and modified configuration information 124 include application configuration information 140 and modified application configuration information 144, respectively. Both application configurations 140 and 144 comprise information used to execute applications on the server 106.

[0040] FIG. 1C is a block diagram illustrating an architecture of a server 106 on which aspects of the present invention may be implemented. For example, the server 106 may be an application server such as an iPlanet™ Application Server, or a web server such as an iPlanet™ Web Server, both available from Sun Microsystems, Inc. of Palo Alto, California. The server 106 includes a container 114, which provides a runtime environment for executing application components. A virtual server 160 is typically implemented to run applications for a specific customer or user, or to support a specific e-commerce or application web site, and operates as a configurable, stand-alone server from the user's viewpoint. Each virtual server 160 can interoperate with more than one application context manager 162, whereby each application context manager 162 facilitates the execution of a single application.

[0041] Container 114 at times contains applications which are running therein, such as applications 150(1) through 150(m) (sometimes referred to collectively herein as applications 150). Applications 150 are any applications that are configured to run on server 106. For example, applications 150 may be servlets using Java™ Servlet technology, which are platform-independent classes in the Java programming language compiled to a byte-code that can be dynamically loaded into and run by a server such as server 106. Servlets are managed by the container 114 to generate dynamic content in

response to client user interaction therewith. The user interactions are referred to as application service requests, or simply service requests or requests. Although embodiments of the invention are described herein as implementations using servlets as application components, practice of the invention is not limited to use with servlets, for the techniques described may apply to other types of server-based applications.

[0042] Container 114, in conjunction with server 106, provides the services over which application requests are met and responses provided. Additionally, container 114 contains and manages application components, such as servlets. Container 114 can be built into a server 106, such as a web server or application server, or installed as an add-on component to a server via an API (Application Program Interface). In one embodiment, container 114 supports HTTP as a request and response protocol, and may additionally support other protocols such as HTTPS (HTTP over SSL). Container 114 provides an environment for running applications, that is, it provides the infrastructure, resources, and external resource interface modules that an application requires to execute properly. For example, the container 114 can load the applicable application classes for instantiating and executing a requested application service; it can load database drivers upon an application making a database service request; and it can maintain and administer user sessions during execution of an application. Container 114 may be embodied in a module such as a Java Virtual Machine, but it is not so limited.

[0043] Once a user requests an application service, typically by invoking a Uniform Resource Locator (URL) through a web browser program, the server 106 identifies the requested application by mapping the URL to an application context. For example, consider the following URL: http://server.com/catalog/shoppingservlet. The

"server.com" portion of the URL maps to a virtual server 160. The "catalog" portion of the URL is a context path which maps to an application named "catalog." The "shoppingservlet" portion of the URL is a URI (Uniform Resource Identifier), and maps to an application component named "shoppingservlet." Once the server 106 resolves the URL to the appropriate application context, it can direct the container 114 to service the request, i.e., to invoke the application component identified in the URI. Thus, from the mapped request passed from the server 106, the application context manager 162 has the necessary information to deploy an instance of the application.

[0044] In general, at the server 106, a listener thread receives a service request, and establishes a connection with a container 114. The listener thread queues the connections, with associated runtime configuration data structures 116 (which may be encapsulated in a configuration object embedded in a server connection object), for passing to worker threads that service the connection. The worker threads are capable of processing whatever requests, events, and activities are received on those connections, and utilize the connection object and the associated configuration object for processing. For example, a worker thread works with the server 106 to identify to which virtual server a particular service request belongs, and works with the container 114 to invoke the application service logic, which in turn determines whether the applicable application components (e.g., servlets) are loaded and whether the application class files are current or require reloading. The term thread is used herein to describe the processing resources used to execute a series of instructions embodied in software code.

[0045] As part of a server 106 start-up process, each container 114 associated with a server 106 instance is initialized. As part of the container 114 initialization process, for

each virtual server 160 associated with a particular container 114, a list of applications is accessed to determine whether the virtual server 160 has any applications 150 configured thereon. For each application 150 found on the list, an application-specific configuration information file is read to determine the current configuration for the application 150. Furthermore, the applications 150 are loaded into the container 114. If the applications 150 load successfully, then a server 106 configuration object (e.g., data structures 116) is denoted as the current configuration, to which subsequent service requests will reference. As part of a server 106 reconfiguration process, the container 114 accesses the list of applications 150 and associated current application configuration information files that are constituent to a new server 106 configuration, which may contain application configurations for new versions of applications. Through the server reconfiguration process, applications 150 executing thereon are reconfigured to their latest version, without requiring a server 106 restart, without interrupting any pending application service requests, and without requiring establishment of any additional user sessions.

[0046] APPLICATION CONFIGURATION

[0047] Applications that execute on a server, such as applications 150, are often referred to as web applications in the context of a WAN such as the Internet, and can be referred to as network or distributed applications in the context of a LAN such as an intranet or enterprise network. Herein, such an application will be referred to simply as "application." An application 150 may comprise a collection of servlets, utility classes, JavaServer Pages™ (JSP), HTML pages, GIFs (Graphics Interchange Format), and other resources that can be run in multiple containers 114 on multiple servers 106. An application 150 is rooted at a specific context path within a server 106, through which all

requests appropriately referencing the path will be routed in order to request the service offered by the application 150.

[0048] A JSP utilizes an extension of the servlet technology to support authoring of HTML and XML pages, making it easier to combine fixed or static template data with dynamic content. JSPs are often used as the front end to a back end servlet, both running on a server such as server 106, to present information to a user wherein the dynamic information is typically provided by the servlet.

[0049] An application 150 exists as a hierarchically organized set of directories. A special directory, named "WEB-INF", exists within the application hierarchy and contains all things related to the application that are not in the application document root. Furthermore, no files in the WEB-INF directory are part of the public document tree, thus they can not be served directly to a client. A WEB-INF directory comprises the following:

[0050] (1) a deployment descriptor (web.xml), which is read by a container 114 for loading the appropriate servlets and executing the application;

[0051] (2) a class directory (classes/*) for servlet and utility classes, which is used by an application class loader to load classes; and

[0052] (3) a library of archive files (lib/*.jar) for Java Archive files that contain servlets, beans, and other utility classes useful to the application 150, the files which are used by the application class loader to load classes. Applications 150 can be packaged into a Web Archive (.war) format.

[0053] The following types of configuration and deployment information typically exist in the application deployment descriptor, but are not limited thereto:

[0054] (1) servletContext initialization parameters, which declare the application's servlet context initialization parameters and associated values;

[0055] (2) session configuration, which defines the session parameters for the application;

[0056] (3) servlet/jsp definitions, which declare servlet data, such as the canonical name and the fully qualified class name of the servlets;

[0057] (4) servlet/jsp mappings, which define mappings between servlets and a URL pattern;

[0058] (5) MIME type mappings, which define mappings between an extension (e.g., "txt") and a MIME type (e.g., "text/plain");

[0059] (6) Welcome file list, which contains an ordered list of welcome file elements;

[0060] (7) error pages, which defines mappings between error codes or exception types to the path of an application resource; and

[0061] (8) security, which defines various security configurations.

[0062] These and other elements of an application deployment descriptor are described in Java™ Servlet Specification (Version 2.2), published by Sun Microsystems, Inc., which are included as part of a configuration object utilized by the container 114 to execute an application.

[0063] Generally, an application developer may modify the application code, configuration, or external class references in reconfiguring an application. Furthermore, any of the aforementioned information elements, among others, can be modified by an application developer, thus reconfiguring an application and necessitating a server

reconfiguration operation in order to implement the changes on the server on which the application executes. In addition, a change to the container 114 in which an application runs can also trigger a server reconfiguration operation in order to reconfigure the container 114 as well as the applications 150 which rely on the container 114 for execution. Any changes to static content, such as GIFs or HTML pages, do not trigger a reconfiguration operation.

[0064] APPLICATION RECONFIGURATION PROCESS

[0065] In general, the server-based application reconfiguration techniques described herein accept a reconfiguration request, build in the background new application configuration files based on new application configuration information, exchange configuration files at an appropriate time, and manage multiple configurations such that pending requests on existing connections are completed using old configurations on which they previously relied and new requests on new connections are serviced using the new configuration.

[0066] Typically, when a server 106 is initializing either at startup or reconfiguration, the server 106 reads main configuration files and builds lists of ports, virtual servers, SSL certificates, and the like. The reconfiguration process can be initiated with a command or message sent to the server 106 across a socket specified as an administration channel, where it is received and processed by a listener thread. In an implementation in which the server is running a UNIX operating system, a UNIX domain socket is used. In an implementation in which the server is running Microsoft NT, a service channel is used. In each case, the reconfiguration message is received by the server without impacting or

interrupting current processing. While building the configuration, the server 106 reads document directories, including application directories such as WEB-INF.

[0067] As mentioned, an application 150 can be reconfigured on a server 106 through initialization of a container 114, which in one embodiment, is a sub-process to the overall server 106 reconfiguration process described above and in U.S. Patent Application No. 09/792,805 entitled "Mechanism for Reconfiguring A Server Without Incurring Server Down Time."

[0068] FIGs. 2A and 2B are flowcharts illustrating a method for reconfiguring an application that executes on a server or other computer system without restarting the server/computer system.

[0069] At step 202, application configuration information defining a reconfigured version of an application 150 is read from storage. This process can be initiated, for example, by a server reconfiguration request or an application reconfiguration request received over an administration channel or via a command line utility. In addition, an application reconfiguration request can be made and processed without any changes to the server 106 configuration, that is with application changes only. In one embodiment, during installation and initialization of an application in container 114, a server reconfiguration request can be propagated up to the server processes from the container 114, essentially requesting the server 106 reconfigure itself. Resultantly, the container 114 and the application 150 are reconfigured. This reconfiguration request process facilitates monitoring the impact of installing a new application configuration on an operating server 106.

[0070] The server 106 can also be reconfigured repeatedly and in quick succession, without affecting the functionality of the server 106. The server 106 monitors and maintains older server configurations, including application configurations, that are processing pending requests and does not delete an older configuration until all request processing relying on the older configuration has completed.

[0071] The application configuration information that defines an application includes at least some of the information in the WEB-INF file (or a similar file), primarily from the web.xml file, as described above. Other information contributing to proper execution of an application, not described above, may also be considered configuration information and still fall within the scope of the present invention. A configuration object encapsulating the configuration generally includes the application classes and other resources, or at least directions to the resources, that an application needs to execute properly.

[0072] An application configuration based on the application configuration information is constructed, at block 204. Each application configuration is associated with at least one server 106 configuration object. Note that an application configuration may be associated with more than one server 106 configuration object. The server 106 is capable of detecting changes to application configuration files, such as those in WEB-INF. For example, the server 106 can read the timestamps associated with application configuration files as stored, to determine, by comparing with other timestamps stored in the current server 106 configuration, whether the stored application is different than the version of the application as currently configured in the server 106 configuration object. Furthermore, the configuration attributes of an application are constructed based on

values in an application configuration file as well as based on the configuration of the container 114. That is, changes to a container 114 in which an application runs inherently also affect the configuration of the application itself. In one embodiment, changes to both the application 150 and the container 114 are considered when constructing an application configuration.

[0073] In a related embodiment, a mechanism is provided for detecting when application class files are revised. The last modified time of each class file in an application configuration is stored and when a request for a web application is received, the last modified time is compared to the timestamp stored in the application configuration. If the last modified timestamp is more recent than the configuration timestamp, then the class file is reloaded into the container 114 for execution.

[0074] Referring back to FIG. 2A, at block 206, after constructing an application configuration based on the reconfigured version of the application, the server 106 (FIG. 1) determines whether the reconfigured version of that application successfully initialized. The server 106 relies on its components, such as container 114, to determine that the reconfigured version of the application successfully initialized since the container 114 reads the application configuration and loads the application. Note that the server 106 has not yet changed the current configuration pointer 112 (FIG. 1A) to reference the new application configuration, and does not change the pointer 112 to reference the new configuration until its initialization has been verified. Thus, new application service requests are not directed to the new configuration until it is determined that the application initialized properly, so all pending requests continue to be serviced using the current application configuration.

[0075] If the reconfigured application results in errors in the container 114, the reconfiguration is rejected and consequently the application is not installed in the server 106 and the error is not propagated throughout the server 106. This is advantageous in that a situation in which an application does not operate properly and thus stops processing, due to installation of corrupted files, is avoided. Furthermore, this feature avoids having to restart and reinitialize the server 106 and reinitialize the container 114 due to corrupted application or application configuration files. Note that the particular application configuration associated with the application with errors is rejected, but other applications that may be reconfigured as part of the same server 106 reconfiguration process may still be installed if no related errors are encountered.

[0076] At block 208, the application configuration is provided to server 106 or other computer system on which the application is installed, for servicing new application service requests coming from new connections, while maintaining one or more old application configurations for servicing pending service requests from existing connections. As such, during a server 106 reconfiguration operation, the existing application objects and their associated configuration objects are not changed. Rather, a separate set of objects is constructed to represent the reconfigured version of the application and its configuration. Once the new configuration is ready to be installed in the running server 106, it is defined as the "current" configuration, while the configuration that it replaced is marked as "old." This occurs through redirecting the current configuration pointer 112 to point to the current configuration. Note that web browsers differ as to whether they maintain a connection for multiple client requests. For a browser that maintains a single connection throughout a user session, the new

configuration would not be utilized for that session, and new requests would be processed referencing the old configuration even after the pointer 112 is redirected. For a browser that establishes a new connection for each request of a user session, the new configuration would be utilized upon establishment of the next new connection pursuant to a user service request subsequent to the pointer redirection.

[0077] In one embodiment, messages related to the initialization of the new configuration are logged, so that a user can track any problems that may occur during the application reconfiguration process. In another embodiment, the logged messages are displayed on a display device so that a user can track any problems in real-time while the reconfiguration process is executing.

[0078] At block 210 of FIG. 2B, persistent session information related to existing user sessions is accessed so that new service requests from the same user and for the same application can use the existing session information. The session information is "copied" to, or associated with, the new version of the application through a pointer that references the area of memory on which the session information is stored. Thus, a new session need not be started, and a user does not have to redo whatever has already been done. For example, if an e-commerce "shopping cart" had items in it prior to an application reconfiguration, the shopping cart does not need to be reloaded with the items because the session information is stored in a persistent manner, for example, on a disk or in a database. Therefore, data consistency is maintained across application reconfigurations.

[0079] Finally, at block 212, old application configurations are destroyed upon completion of all service requests that are using a particular old configuration, thereby freeing computing resources. For example, the memory blocks which are used to store

the old configuration are released for other uses. In one embodiment, determining whether a particular application configuration is referenced by a pending request comprises determining whether a reference count associated with the old server configuration is equal to a predetermined value. For example, a counter can be maintained for a server configuration whereby each request referencing that configuration causes the counter to be incremented, whereas completion of each request referencing that configuration causes the counter to be decremented. Therefore, the counter reaching zero would indicate that no connections or processes are currently referencing the configuration, and consequently no application service requests, and the configuration can be deleted. The example presented is a simplified example, for other actions may affect the counter in addition to requests started and requests completed.

[0080] Note that the lifecycle of the application is tied to the underlying server configuration. Hence, when a server is reconfigured, it has associated with it a set of applications, and although the applications may be identical to the previous versions of the applications, the memory used to store the revised application configurations is separate from the memory used to store the previous or old application configurations.

[0081] CONFIGURATION MANAGER

[0082] As noted previously, it is the configuration manager 110 (FIG. 1A) that is responsible for constructing and maintaining the configuration data structures 116 (FIG. 1A) and the current configuration pointer 112 (FIG. 1A). Operation of the configuration manager 116 begins at system startup. More specifically, the configuration manager 110 is instantiated at startup time, and once instantiated, it proceeds to construct an initial set of configuration data structures 116(1). In doing so, the configuration manager 110

accesses a set of configuration information 120 (FIG. 1B), including application configuration information 140 (FIG. 1B), from a storage 118 (FIG. 1B), and transforms the configuration information into a set of internal data structures 116(1), including application configuration 146(1) (FIG. 1A). These data structures 116(1) are basically internal server representations of the configuration information 120, which are preprocessed to facilitate access at runtime by various components of the server 106, including the request processing module 108 and the container 114 (FIG. 1A). The manner in which the data structures 116(1) is constructed, and the form that they take within the server 106 will differ from implementation to implementation. As the data structures 116(1) are constructed, they are stored within a portion of the server's memory space. Once construction of the data structures 116(1) is completed and verified, the configuration manager 110 updates the current configuration pointer 112 (which in one embodiment takes the form of a global variable) to point to the memory space occupied by the configuration data structures 116(1), thereby declaring the data structures 116(1) to be the current server configuration. In addition, the configuration manager 110 increments the reference count 130(1) (FIG. 1A) of the data structures 116(1). Once that is done, the server 106 is configured and is ready for operation. At that point, the request processing module 108 may begin establishing connections and servicing requests on those connections using the current configuration pointer 112 and the configuration data structures 116(1).

[0083] After the initial set of configuration data structures 116(1) is constructed, the configuration manager 110 enters a monitoring mode wherein it monitors for a signal indicating that the server 106 is to be reconfigured. If no such signal is detected, then the

configuration manger 110 loops back to continue checking for the signal. However, if such a signal (e.g. a Unix signal) is received, then the configuration manager 110 proceeds to construct a new set of configuration data structures. In one embodiment, when the reconfiguration signal is sent to the configuration manager 110, a reference to a modified set of configuration information 124 (FIG. 1B), including modified application configuration information 144 (FIG. 1B), is also provided. Using the reference to access the modified configuration information 124 stored in the storage 118, the configuration manager 110 constructs a new set of configuration data structures 116(2) (FIG. 1A) based upon the modified configuration information 124. In one embodiment, the configuration manager 110 constructs the new data structures 116(2) in much the same way as it did the initial set of data structures 116(1), except that the new data structures 116(2) are not constructed as part of a startup procedure.

[0084] As the new configuration data structures 116(2) are constructed, they are stored into another portion of the server's memory space. While the new configuration data structures 116(2) are being constructed, the current configuration pointer 112 is not changed. That is, it still points to the initial set of configuration data structures 116(1). As a result, all connections established by the request processing module 108 and all application service requests during construction of the new data structures 116(2) will be associated with the initial set of data structures 116(1), not the new set 116(2).

[0085] After the new configuration data structures 116(2) are completely constructed, the configuration manager 110 updates the current configuration pointer 112 to point to the new configuration data structures 116(2). In one embodiment, this update is performed atomically (for example, by using a read/write lock) to prevent other threads

from accessing the pointer 112 while it is being updated. After the pointer 112 is updated, the reference count 130(2) (FIG. 1A) of the new data structures 116(2) is incremented, and the reference count 130(1) of the initial set of data structures 116(1) is decremented. Thereafter, the configuration manager 110 checks the value of the reference count 130(1). If the reference count 130(1) indicates that no more connections are relying upon the initial set of data structures 116(1), then those data structures 116(1) are released or destroyed. In one embodiment, the configuration manager 110 releases or destroys the data structures 116(1) by freeing the memory space occupied thereby.

[0086] Once the current configuration pointer 112 is updated, the new configuration data structures 116(2) are established as the current server configuration, and are used by the various server 106 components to govern their behavior, and the application configurations 146 are used by the container 114 to govern execution of the applications 150. In this manner, the configuration of the server 106 is changed from one configuration 116(1) to another 116(2) without shutting down and restarting the server 106. Hence, the configuration of any revised applications 150 (FIG. 1A) are also changed from application configuration 146(1) to 146(2). After the change in configuration is completed, the request processing module 108 will associate new connections with the new set of configuration data structures 116(2) rather than the old set 116(1). After the server 106 is reconfigured, the configuration manager 110 loops back to monitor for another reconfiguration signal. In the manner described, the configuration manager 110 reconfigures the server 106, and thus the applications 150, smoothly, efficiently, and without incurring any server down time.

[0087] HARDWARE OVERVIEW

[0088] In one embodiment, the server 106 of the present invention and its various components are implemented as a set of instructions executable by one or more processors. The invention may be implemented as part of an object oriented programming system, including but not limited to the JAVA™ programming system manufactured by Sun Microsystems, Inc. of Palo Alto, California.

[0089] Fig. 3 shows a hardware block diagram of a computer system 300 in which an embodiment of the invention may be implemented. Computer system 300 includes a bus 302 or other communication mechanism for communicating information, and a processor 304 coupled with bus 302 for processing information. Computer system 300 also includes a main memory 306, such as a random access memory ("RAM") or other dynamic storage device, coupled to bus 302 for storing information and instructions to be executed by processor 304. Main memory 306 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 304. Computer system 300 further includes a read only memory ("ROM") 308 or other static storage device coupled to bus 302 for storing static information and instructions for processor 304. A storage device 310, such as a magnetic disk, optical disk, or magneto-optical disk, is provided and coupled to bus 1002 for storing information and instructions.

[0090] Computer system 300 may be coupled via bus 302 to a display 312, such as a cathode ray tube ("CRT") or a liquid crystal display ("LCD"), for displaying information to a computer user. An input device 314, including alphanumeric and other keys, is coupled to bus 302 for communicating information and command selections to processor

304. Another type of user input device is cursor control 316, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 304 and for controlling cursor movement on display 312. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0091] The invention is related to the use of computer system 300 for dynamically reconfiguring an application. According to embodiments of the invention, dynamically reconfiguring an application is provided by computer system 300 in response to processor 304 executing one or more sequences of one or more instructions contained in main memory 306. Such instructions may be read into main memory 306 from another computer-readable medium, such as storage device 310. Execution of the sequences of instructions contained in main memory 306 causes processor 304 to perform the process described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0092] The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 304 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical, magnetic, or magneto-optical disks, such as storage device 310. Volatile media includes dynamic memory, such as main memory 306. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 302. Transmission media can

also take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications.

[0093] Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0094] Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 304 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 300 can receive the data on the telephone line and use an infrared transmitter to convert the data to an infrared signal. An infrared detector can receive the data carried in the infrared signal and appropriate circuitry can place the data on bus 302. Bus 302 carries the data to main memory 306, from which processor 304 retrieves and executes the instructions. The instructions received by main memory 306 may optionally be stored on storage device 310 either before or after execution by processor 304.

[0095] Computer system 300 also includes a communication interface 318 coupled to bus 302. Communication interface 318 provides a two-way data communication coupling to a network link 320 that is connected to a local network 322. For example, communication interface 318 may be an integrated services digital network ("ISDN") card

or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 318 may be a local area network ("LAN") card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 318 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0096] Network link 320 typically provides data communication through one or more networks to other data devices. For example, network link 320 may provide a connection through local network 322 to a host computer 324 or to data equipment operated by an Internet Service Provider ("ISP") 326. ISP 326 in turn provides data communication services through the worldwide packet data communication network now commonly referred to as the "Internet" 328. Local network 322 and Internet 328 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 320 and through communication interface 318, which carry the digital data to and from computer system 300, are exemplary forms of carrier waves transporting the information.

[0097] Computer system 300 can send messages and receive data, including program code, through the network(s), network link 320 and communication interface 318. In the Internet example, a server 330 might transmit a requested code for an application program through Internet 328, ISP 326, local network 322 and communication interface 318

[0098] Processor 304 may execute the received code as it is received, and/or stored in storage device 310, or other non-volatile storage for later execution. In this manner, computer system 300 may obtain application code in the form of a carrier wave.

[0099] EXTENSIONS AND ALTERNATIVES

[00100] Alternative embodiments of the invention are described throughout the foregoing description, and in locations that best facilitate understanding the context of the embodiments. Furthermore, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. For example, the invention is described with reference to applications embodied in servlets, with configurations represented in WEB-INF directories, which comprise components including deployment descriptors embodied in web.xml files. However, the techniques described herein are not limited to any specific application architecture, runtime environment, or server platform. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

[00101] In addition, certain process steps are set forth in a particular order, and alphabetic and alphanumeric labels may be used to identify certain steps. Unless specifically stated in the description, embodiments of the invention are not necessarily limited to any particular order of carrying out such steps. In particular, the labels are used merely for convenient identification of steps, and are not intended to specify or require a particular order of carrying out such steps.

[00102] One extension that can be implemented with the system described herein is a mechanism for comparing application configurations, or configuration constituents, to determine if a configuration has been changed, and if so, in what manner.